EIE
Escuela de
Ingeniería Eléctrica

Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

UNIVERSIDAD DE
COSTA RICA

IE-0117
Programación Bajo Plataformas Abiertas

Laboratorio #4

Fernando Jiménez Ureña B74020

II ciclo
Octubre 2021

1. Resumen

En el presente laboratorio se introducirá al lenguaje de programación C, se llevará a cabo una investigación así como la compilación de código dentro del mismo. Además, se investigará sobre programas sumamente útiles cuando de programación se trata como son el caso de Git y Doxygen, y cuyas funciones serán descritas a fondo más adelante.

El desarrollo de este laboratorio consta de 5 problemas que deben de resolverse mediante el lenguaje de programación C y que serán explicados y resueltos más adelante en el Análisis de Resultados.

2. Nota Teórica

Es importante definir qué es un lenguaje de programación para tener un mejor entendimiento de este laboratorio. En general, un lenguaje de programación es un lenguaje formal que mediante una serie de instrucciones, le permite a un programador escribir un conjunto de órdenes, acciones consecutivas, datos y algoritmos para, de esa forma, crear programas que controlen el comportamiento físico y lógico de una máquina. [1]

Existen una amplia variedad de lenguajes de programación, cada uno con ventajas y desventajas así como diferentes funciones. En este caso, se hará del lenguaje de Programación 'C'. Este lenguaje fue creado entre 1969 y 1972 como el sucesor de otro lenguaje de programación que existía en esta época conocido como 'B'. El lenguaje 'C' es sumamente eficiente y es el lenguaje más popular para para crear software de sistemas, y es también muy utilizado para crear aplicaciones. Posee una importante característica ya que dispone de las estructuras típicas de los lenguajes de alto nivel pero, pero a su vez, dispone de construcciones del lenguaje que permiten un control a muy bajo nivel. [2]

Para entender mejor la sintaxis de C, se pueden destacar las siguientes funciones:

- int main() : Es la función primaria de un programa. Es utilizada como el punto de inicio para la ejecución del programa.
- int : Utilizada para definir variables de tipo entero
- float : Utilizada para definir variables de tipo flotante
- scanf() : Se utiliza para introducir cualquier combinación de valores numéricos, caracteres sueltos y cadenas de caracteres a través del teclado hacia el programa.
- printf() : Se utiliza para imprimir datos en el dispositivo de salida estándar (en la pantalla)
- rand() : Genera un número aleatorio
- srand() : Funciona para iniciar el generador de números aleatorios, creando una semilla.
- /n : Con este comando se imprime un salto de línea dentro del comando printf()

La programación se puede complementar con poderosas herramientas como lo son Git y Doxygen. Git es un programa que se utiliza para el control de versiones de código fuente. El control de versiones es la práctica de rastrear y gestionar los cambios en el código de software. Esto ayuda a los equipos de software a gestionar los cambios en el código fuente a lo largo del tiempo, esto les permite trabajar de forma más rápida e inteligente. Git es uno de estos sistemas de control de versiones y es por mucho el más moderno y utilizado del mundo. [3]

Por otro lado existe otra herramienta bastante útil como fue ya mencionada anteriormente: Doxygen. Este es un programa que se encarga de generar la documentación de un proyecto de programación a partir de los archivos con los códigos fuentes del mismo. Es utilizado en conjunto a lenguajes de programación como es el caso de C, Java y C++, entre otros.[4]

Complementar la práctica de programación con las herramientas recién descritas es de gran utilidad para mejorar la productividad así como el desarrollo del código a la hora de trabajar en proyectos especialmente cuando hay varias personas trabajando en el mismo.

3. Análisis de Resultados

Es importante explicar brevemente los códigos desarrollados en el laboratorio en esta sección del informe a pesar de que dichos códigos se encuentran en el repositorio creado en el Git de la Escuela de Ingeniería Eléctrica.

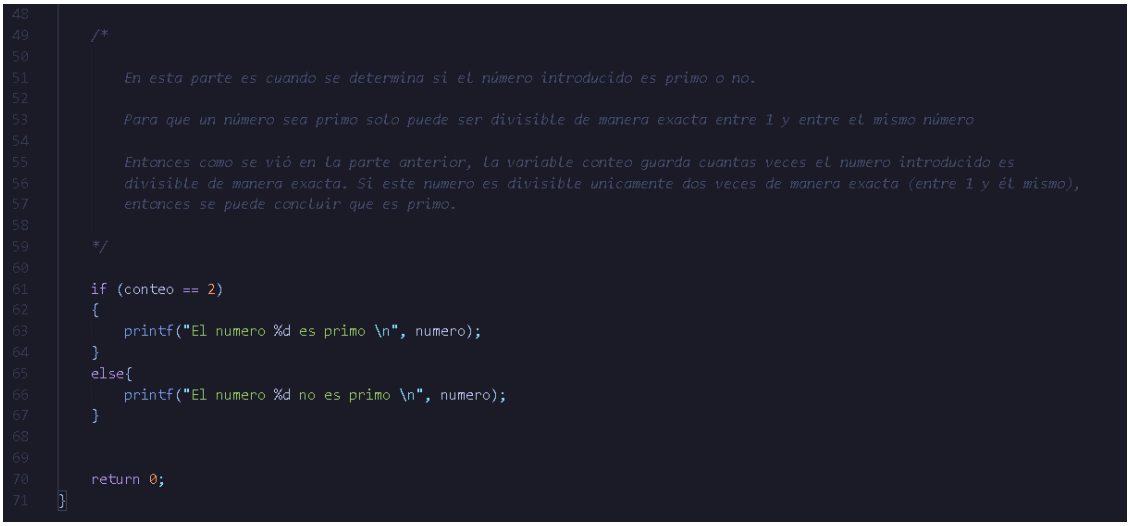
3.1. Números Primos

En este problema se solicita realizar un programa que le solicite al usuario un número y determine si es un número primo o no. Para este código, se utilizaron los condicionales if y else luego de que el programa analice el número introducido. La explicación completa de como fue implementado el algoritmo para la resolución del problema se encuentra documentada dentro del código.

En la Figura 1., Figura 2. y Figura 3. se puede apreciar la implementación y ejecución del código para el problema planteado de determinar si un Número es Primo o no.

```c
#include<stdio.h>

int main(){

    //En este punto se declara la variable "numero" que será el numero que introducirá el usuario

    int numero;

    // Se declara otra funcion llamada conteo para utilizarla dentro del ciclo for creado mas abajo

    int conteo = 0; //Se inicializa en cero ya que el objetivo es ir agregando valores cada vez que se recorre en el ciclo

    // Se utilizan la funcion printf para solicitar en pantalla el numero al usuario
    // Y se utiliza la funcion scanf para guardar dicho valor en la variable "numero"

    printf("Digite un numero para determinar si es un numero primo o no:\n");
    scanf("%d", &numero);

    // Se procede a utilizar un ciclo for. El fin de usar este ciclo es recorrer varias veces el mismo proceso

    //Este ciclo lo que hace es recorrer la instruccion desde 1 hasta el numero introducido.

    /*
        La instruccion lo que hace es dividir el numero introducido entre el valor i que esta recorriendo en ese momento

        El caracter "%" signfica que está calculando el residuo de dicha division. Si el residuo de dicha division es igual a 0
        significa que la division es exacta.

        Entonces la instruccion lo que dicta es que cada vez que el ciclo for encuentre divisiones exactas, vaya sumando esos valores
        y guardandolos dentro de la variable "conteo".

    */

    for (int i = 1; i <= numero; i++)
    {
        if(numero %i == 0){
            conteo++;
        }
    }
```

Figura 1: Código desarrollado para el problema de números primos



```c

    /*

        En esta parte es cuando se determina si el número introducido es primo o no.

        Para que un número sea primo solo puede ser divisible de manera exacta entre 1 y entre el mismo número

        Entonces como se vió en la parte anterior, la variable conteo guarda cuantas veces el numero introducido es
        divisible de manera exacta. Si este numero es divisible unicamente dos veces de manera exacta (entre 1 y él mismo),
        entonces se puede concluir que es primo.

    */

    if (conteo == 2)
    {
        printf("El numero %d es primo \n", numero);
    }
    else{
        printf("El numero %d no es primo \n", numero);
    }


    return 0;
}
```

Figura 2: Código desarrollado para el problema de números primos

```
fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Labo 4$ gcc -o exe primos.
c
fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Labo 4$ ./exe
Digite un numero para determinar si es un numero primo o no:
5
El numero 5 es primo
fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Labo 4$ ./exe
Digite un numero para determinar si es un numero primo o no:
8
El numero 8 no es primo
```

Figura 3: Código en ejecución

En la Figura 4. se puede apreciar el Diagrama de Flujos del programa.

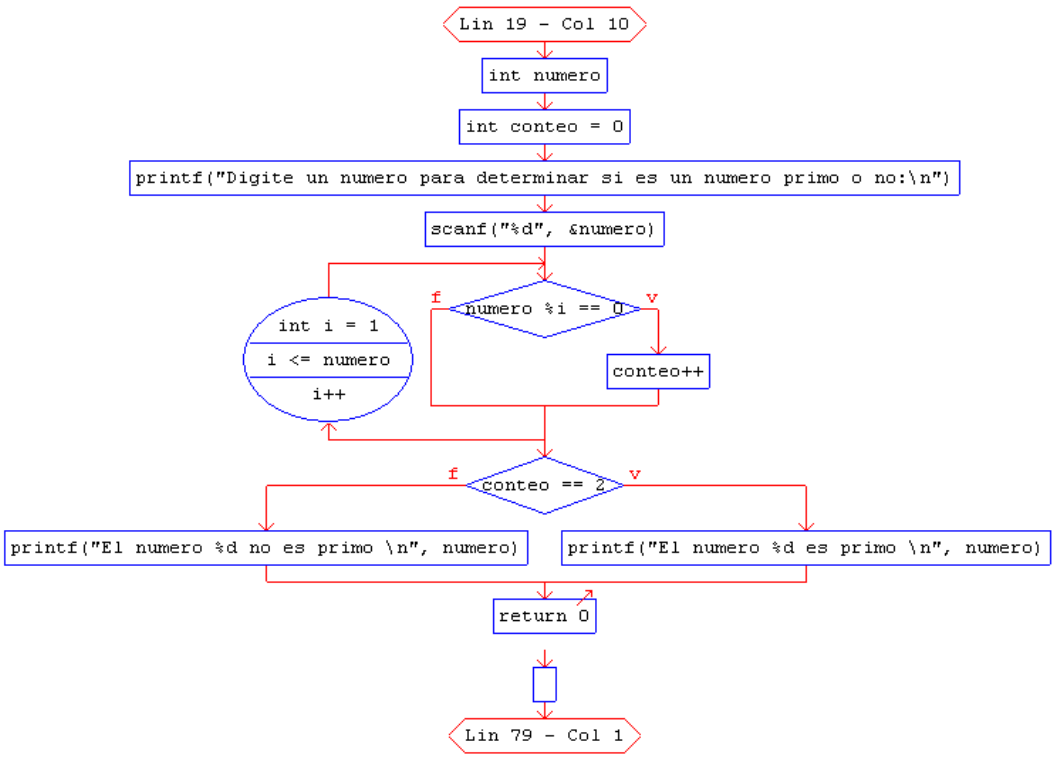


Figura 4: Diagrama de Flujo para el problema de números primos

3.2. Suma de Variables Aleatorias

Seguidamente, en este problema se plantea que se sumen 1000 valores pares aleatorios entre dos límites que se le solicitan al usuario. Para este problema, se acudió al uso de bucles *for* para recorrer el intervalo que definió el usuario. Sin embargo lo más destacado de este problema fue el uso de la función *srand()* que se utiliza para generar números aleatorios.

En la Figura 5., Figura 6., Figura 7.y Figura 8. se muestra el código implementado y en ejecución. Cabe destacar que dentro del código fuente se explica de manera detallada el algoritmo utilizado para resolver problema.

```c
#include<stdio.h>
#include<stdlib.h> //aqui esta el generador de numeros aleatorios
#include<time.h>
#include <stdio.h>

// Se agregan nuevas bibliotecas que se utilizan para poder generar los numeros aleatorios

int main(){

    /*

        Primero se declaran las variables que se utilizaran para desarrollar el programa

        En la variable aleatorio se almacenara un numero aleatorio que sera creado haciendo uso de una libreria

        Los limites superiores e inferiores se le solicitan al usuario para determinar entre cuales valores se creará
        el numero aleatorio.

        La variable suma guarda los valores de los numeros aleatorios generados y los suma

    */

    int aleatorio, limite_inferior,limite_superior, suma = 0;

    printf("Digite un limite inferior:\n");
    scanf( "%d" , &limite_inferior);

    printf("Digite un limite superior:\n");
    scanf( "%d" , &limite_superior);


    srand(time(NULL)); //Este comando genera el numero aleatorio
```

Figura 5: Código desarrollado para el problema de la Suma de Números Pares Aleatorios

```c
    /*

        Como se deben generar 1000 numeros aleatorios, se procede a utilizar un ciclo for que recorra desde 0 hasta 1000 la misma instruccion

        En este caso, la instruccion dicta que en cada vuelta, se genere un numero par aleatorio entre los los limites definidos por el usuario.

        La formula para generar un numero aleatorio par es:
        aleatorio= ((limite_inferior/2) + rand()%((limite_superior - limite_inferior+2)/2)) * 2;

    */

    for (int i = 1; i <= 1000; i++)
    {
        aleatorio= ((limite_inferior/2) + rand()%((limite_superior - limite_inferior+2)/2)) * 2;

        // Este print se utiliza para mostrar en pantalla los valores generados en cada vuelta

        printf("%d. %i \n", i, aleatorio);

        /*

            Ya una vez generado el numero par aleatorio, dicho valor se almacena en la variable suma.

            Y este valor de suma se va sumando por cada vuelta del ciclo hasta sumar los 1000 valores generados

        */
        suma += aleatorio;

    }

    // Este printf imprime en pantalla la suma total de los 1000 valores pares generados.

    printf("La suma de los valores es: %d\n",suma);

    return 0;
}
```

Figura 6: Código desarrollado para el problema de la Suma de Números Pares Aleatorios

```
fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Labo 4$ gcc -o exe suma.c
fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Labo 4$ ./exe
Digite un limite inferior:
10
Digite un limite superior:
523
1. 294
2. 60
3. 318
4. 84
5. 306
6. 20
7. 222
8. 344
9. 348
10. 230
11. 334
12. 336
13. 82
14. 308
15. 264
16. 478
17. 486
18. 272
19. 362
20. 470
21. 174
22. 178
23. 328
24. 454
```

Figura 7: Código en ejecución

```
973. 122
974. 70
975. 20
976. 56
977. 254
978. 486
979. 508
980. 520
981. 168
982. 182
983. 204
984. 422
985. 452
986. 470
987. 90
988. 450
989. 388
990. 300
991. 18
992. 352
993. 62
994. 252
995. 82
996. 118
997. 80
998. 164
999. 498
1000. 408
La suma de los valores es: 263960
```

Figura 8: Código en ejecución

En la Figura 9. se puede apreciar el Diagrama de Flujos del programa.

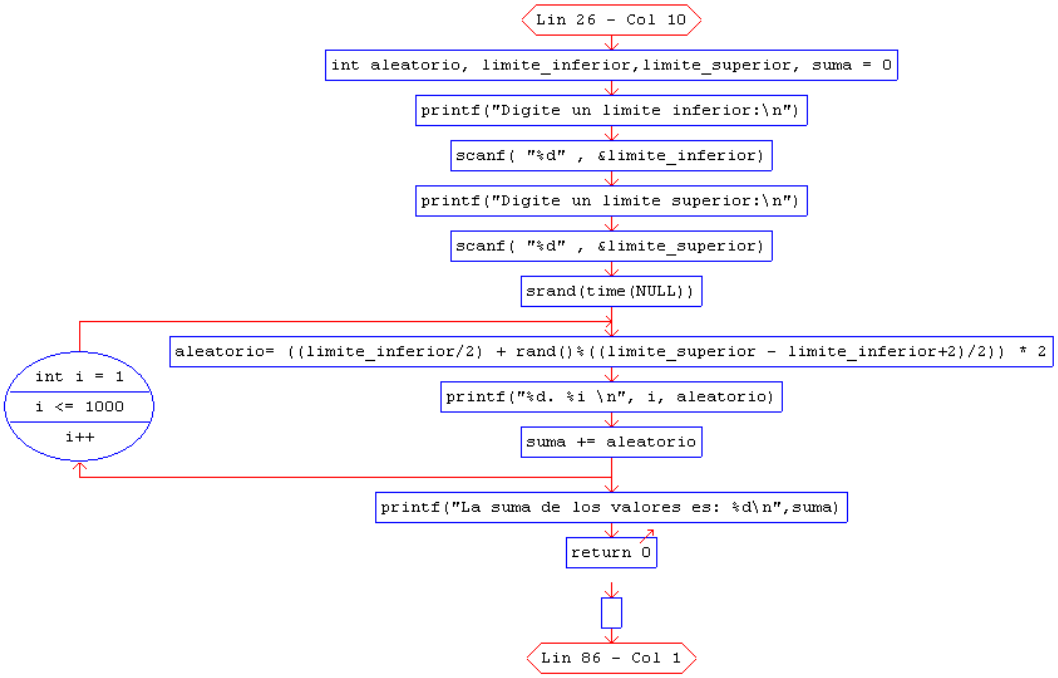


Figura 9: Diagrama de Flujo para el problema de la Suma de Números Pares Aleatorios

3.3. Impresión de Números

Para este problema se solicitaba imprimir en pantalla una pirámide de números de forma iterativa. El programa debía solicitar al usuario la cantidad de filas que deseaba para la pirámide. Para este problema fueron de uso esencial los ciclos *for*. En la Figura 10., Figura 11., y Figura 12. se puede apreciar la ejecución del código y como fue escrito el algoritmo del mismo para poder resolver el problema.

```c
#include <stdio.h>

int main(){

    /*
        Se procede a definir las variables necesarias para poder correr el programa

        En la variable "n" se le solicita y se guardan las cantidad de filas que el usuario desea que se imprima la piramide
    */

    int n, contador1 = 0, contador2 = 0, k = 0;

    printf( "Digite la cantidad de filas que desea que tenga la piramide: ");
    scanf("%d", &n);

    /*
        En este punto, se utiliza un bucle for que recorre las n filas que fueron introducidas por el usaurio.

        Hay otro bulce for anidado que imprime en pantalla la n cantidad  de espacios en la pirámide
    */

    for(int i = 1; i <= n; ++i)
    {
        for(int espacios = 1; espacios <= n-i; espacios++)
        {
            printf("  ");
            ++contador1;
        }

        /*
            Dentro del primer bucle for se añade un condicional while que señala que se cumplan dos condiciones
            mientras el valor de k sea diferente a 2*i-1.
        */

        while(k != 2*i-1)
        {
```

Figura 10: Código desarrollado para el problema de la Impresión de Números

```c
            /*
                Depende de la condicion que se esté cumpliendo en ese momento, se van sumando valores ya sea al contador1 o al contador2
            */

            if (contador1 <= n-1)
            {
                printf("%d", i+k); printf(" ");
                contador1++;
            }
            else
            {
                contador2++;
                printf( "%d" , i+k-2*contador2); printf (" ");
            }
            k++;
        }

        contador2 = contador1 = k = 0;

        printf("\n");

    }

    return 0;
}
```

Figura 11: Código desarrollado para el problema de la Impresión de Números

```
fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Labo 4$ gcc -o exe piramide.c

fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Labo 4$ ./exe
Digite la cantidad de filas que desea que tenga la piramide: 5
        1
      2 3 2
    3 4 5 4 3
  4 5 6 7 6 5 4
5 6 7 8 9 8 7 6 5
```

Figura 12: Código en ejecución

En la Figura 13. se puede apreciar el Diagrama de Flujos del programa.

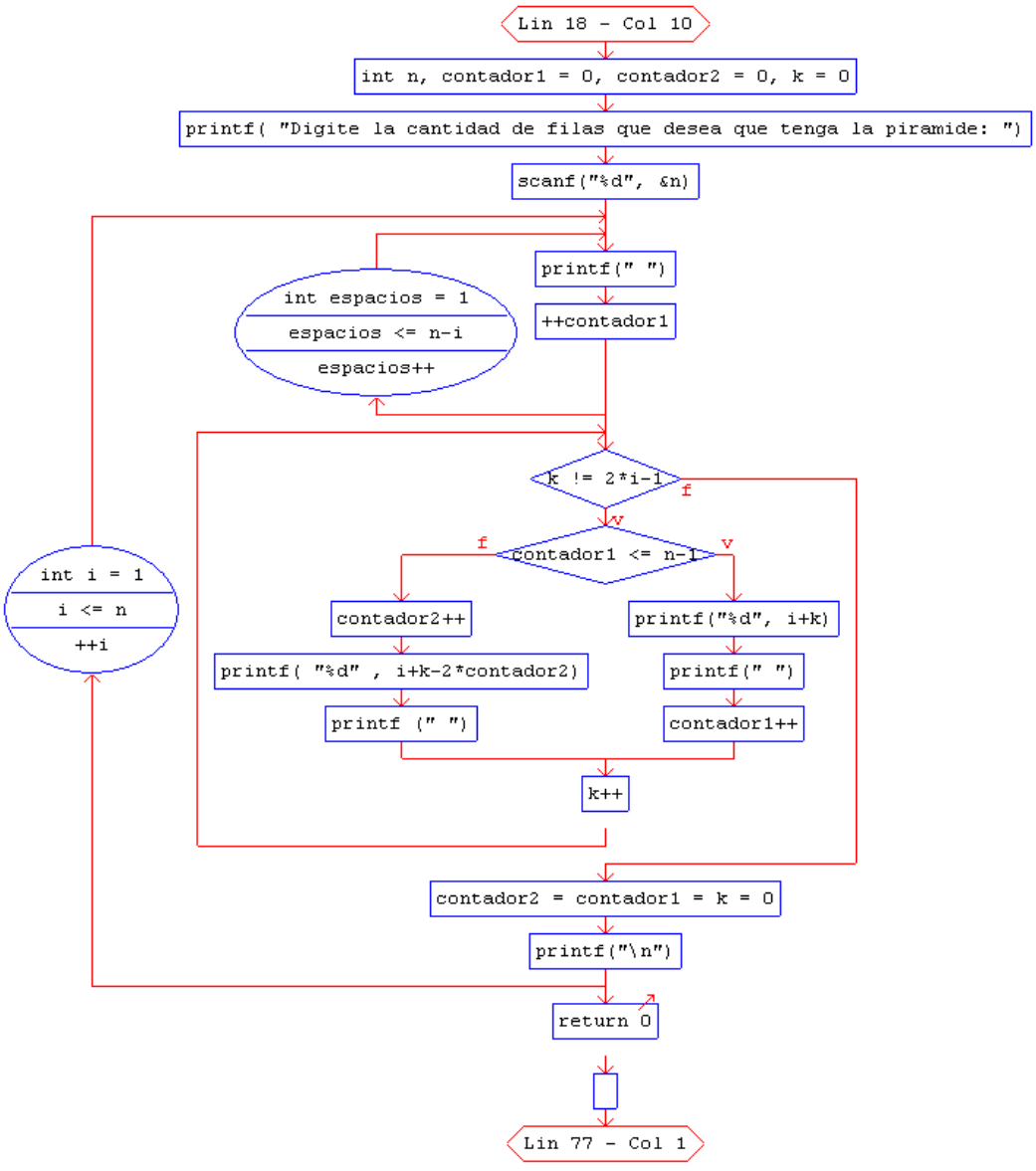


Figura 13: Diagrama de flujos para el problema de la Impresión de Números

3.4. Contar tipo de Caracteres

Luego, para este problema se solicita realizar un programa que cuente la cantidad de vocales, consonantes y dígitos contenidos en una cadena de texto menor a 20 caracteres. Para este caso, se procedió a utilizar otro tipo de ciclo llamado el ciclo *switch*. Este ciclo se encarga de analizar diferentes casos en cada vuelta, y si en cierta vuelta se cumple el caso entonces se suma a uno de los contadores definidos al inicio del programa. Seguidamente, estos contadores se suman para encontrar la cantidad de vocales, consonantes y dígitos totales en la linea de caracteres.

En la Figura 14., Figura 15., Figura 16. y Figura 17. se observa la implementación del código y la ejecución del mismo.

```c
#include<stdio.h>
#include<string.h> //Esta biblioteca se utiliza cuando se desean utilizar cadenas de caracteres

int main(){

    /*

        Se declara la variable caracteres[20] que es la variable que guarda la cadena de 20 caracteres solicitada al usuario

        Ademas, se definen distintos contadores que se utilizaran en el bucle swtich que se explica más adelante

        Finalmente se declaran dos variables más (sumaDeDigitos y sumaDeDigitos_espacios) que es donde se realizan la sumatoria final
        de los casos

    */

    char caracteres[20];

    int vocal_a=0,vocal_e=0,vocal_i=0,vocal_o=0,vocal_u=0, consonante=0, espacio = 0;

    int conteoVocales = 0;

    int sumaDeDigitos = 0, sumaDeDigitos_espacios = 0;

    //Se procede a pedir al usuario el string que desea analizar
    printf("**Atención: note que el programa solo detectará 20 caracteres como máximo de la línea introducida**");
    printf("\n\n");
    printf("Digite una linea de caracteres menor o igual a 20 caracteres: \n");

    fgets(caracteres, sizeof(caracteres), stdin);
    printf("\n");

    /*

        Este printf y el puts le imprime en pantalla al usuario la linea de texto que va a
        analizarse para asi poder confirmar que solo se analizaran los primeros 20 caracteres

    */
```

Figura 14: Código desarrollado para el problema de Contar el tipo de caracteres

```
    printf( "El string leido por el programa es: \n"); puts(caracteres);

    printf("\n");

    /*

        Para poder cumplir la funcion deseada, se procedio a utilizar el bucle switch. En este caso, lo que hace dicho bucle es analizar diferentes
        casos, y cada vez que se cumpla un caso, se irá añadiendo al respectivo contador que fue definido anteriormente.

        En este caso, se tomaron los casos en que cuando se recorra el string, se sumen las diferentes vocales y espacios. En caso de que el string
        no detecte ninguna vocal o espacio, entonces asumirá que es un consonante e irá añadiendo dichos valores a los respectivos contadores.

    */

    for (int i = 0; i < caracteres[i] ; i++){
        switch (caracteres[i])
    {
        case 'a': vocal_a++; break;
        case 'e': vocal_e++; break;
        case 'i': vocal_i++; break;
        case 'o': vocal_o++; break;
        case 'u': vocal_u++; break;
        case 'A': vocal_a++; break;
        case 'E': vocal_e++; break;
        case 'I': vocal_i++; break;
        case 'O': vocal_o++; break;
        case 'U': vocal_u++; break;
        case ' ': espacio++; break;
        default: consonante++;break;

    }
    }
```

Figura 15: Código desarrollado para el problema de Contar el tipo de caracteres

```
    /*

        Luego de recorrer todo el string y almacenar los valores en los contadores, se procede a sumar la cantidad de vocales
        y la cantidad de consonantes y almacenar dichos valores en nuevas variables.

        Tambien se suman las consonantes y vocales y los espacios para determinar la cantidad de caracteres totales en el string
        introducido

    */

    conteoVocales = vocal_a + vocal_e + vocal_i + vocal_o + vocal_u;
    sumaDeDigitos = consonante + conteoVocales;
    sumaDeDigitos_espacios = consonante + conteoVocales + espacio;


    /*

        Finalmente aqui se imprimen en pantalla los distintos resultados obtenidos

    */

    printf("Total de vocales: %d \n", conteoVocales );
    printf("Total de consonantes: %d \n", consonante );
    printf("Total de caracteres sin espacios: %d \n", sumaDeDigitos );
    printf("Total de caracteres con espacios: %d \n", sumaDeDigitos_espacios );
    printf("\n");

    return 0;
}
```

Figura 16: Código desarrollado para el problema de Contar el tipo de caracteres

```
fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Labo 4$ gcc -o exe cadena.c
fer9828@LAPTOP-K1I7TVO4:/mnt/c/Users/jimen/OneDrive/Escritorio/Progra/C/Plataformas 2S - 2021/Labo 4$ ./exe
**Atención: note que el programa solo detectará 20 caracteres como máximo de la línea introducida**

Digite una linea de caracteres menor o igual a 20 caracteres:
Esto es una prueba

El string leido por el programa es:
Esto es una prueba


Total de vocales: 8
Total de consonantes: 7
Total de caracteres sin espacios: 15
Total de caracteres con espacios: 18
```

Figura 17: Código en ejecución

En la Figura 18. se puede apreciar el Diagrama de Flujos del programa.

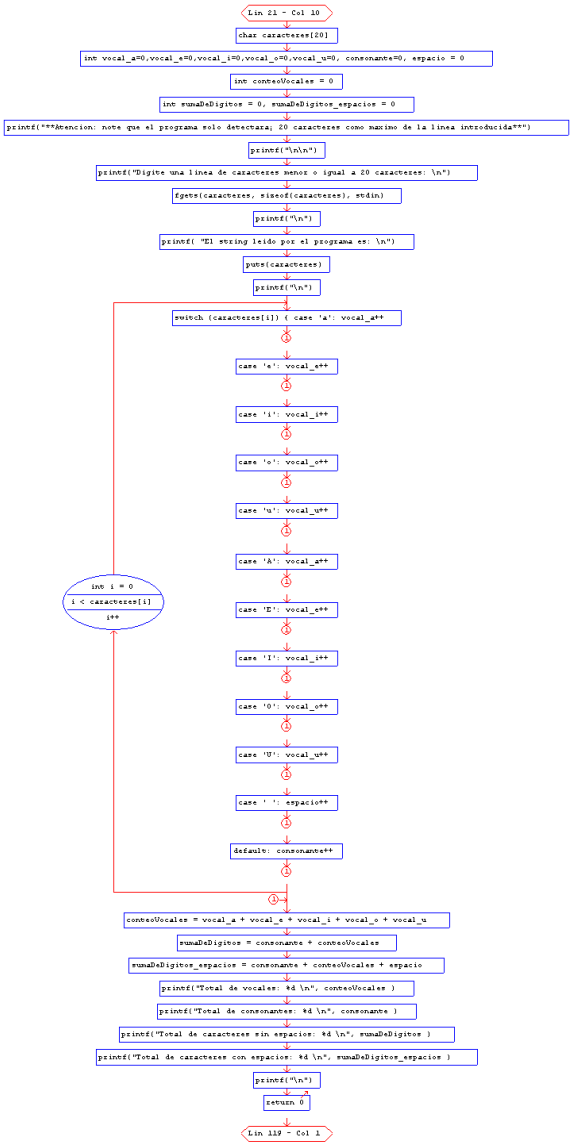


Figura 18: Código desarrollado para el problema de Contar el tipo de caracteres

3.5. Matriz Regular y Determinante

Finalmente, en el último problema se solicita determinar si una matriz es regular mediante el cálculo de su determinante. Para este problema se utilizaron nuevamente los ciclos *for* de manera anidada. Primero para guardar los valores en la matriz y luego para mostrarlos. En la Figura 19., Figura 20. y Figura 21. se encuentra el código desarrollado y ejecutado. Igualmente dentro del código fuente se explica de manera detallada el funcionamiento del algoritmo para la solución del problema.

```c
#include<stdio.h>

int main(){

    /*
        Se define la matriz 3x3 que se estara utilizando

        Ademas de agregar como variables las posiciones a11, a12 y a13 de las matrices ya que aqui se almacenaran
        Los valores de las operaciones que son necesarias para determinar el determinante de una matriz

        Finalmente se define la constante determinante, que sera donde se almacene el valor de la suma de a11, a12 y a13

    */

    int matriz[3][3];
    int a11 = 0, a12 = 0, a13 = 0;
    int determinante = 0;

    //Primero se procede a pedirle al usuario los valores de la matriz para poder relleanrla

    printf("\nRellenando los valores de la matriz: \n\n");
    for (int i = 0; i < 3; i++){ //filas ----
        for (int j = 0; j < 3; j++){ //columnas |||
            printf("Digite un numero\n-> Posicion en la matriz [%d][%i]: ", i+1 , j+1); // Se guarda por posicion. Se agrega el "i+1" y "j+1" para qu
            scanf("%d", &matriz[i][j]);                                                 // en pantalla empiece desde la posicion [1][1]
        }
    }

    // Ya una vez ingresados los valores de la matriz, se procede a mostrar en pantalla la matriz creada para que el usuario
    // pueda observar los valores que ingresó

    printf("\nMostrando la Matriz introducida: \n\n");
    for (int i = 0; i < 3; i++){
        for (int j = 0; j < 3; j++){
            printf(" %d", matriz[i][j]); printf(" ");
        }
        printf("\n");
    }

    printf("\n");
```

Figura 19: Código desarrollado para el problema de Matriz Regular y Determinante

```c
    /*
        Se procede a calcular el determinante de la matriz.

        Recordando que por el metodo de Laplace, se puede calcular el determinante mediante la multpilicacion y suma de las
        distintas posiciones en la matriz.

        Se definieron las variables a11, a12, a13 para realizar estas operaciones y luego se suman sus resultados para obtener el
        determinante de la matriz y se imprime en pantalla el resultado del determinante

    */

    a11 = ((matriz[0][0]) * ((matriz[1][1]*matriz[2][2])-(matriz[1][2]* matriz[2][1])));
    a12 = ((-matriz[0][1]) * ((matriz[1][0]*matriz[2][2])-(matriz[1][2]*matriz[2][0])));
    a13 = ((matriz[0][2]) * ((matriz[1][0]*matriz[2][1])-(matriz[2][0]* matriz[1][1])));

    determinante = a11 + a12 + a13;

    printf("El determinante de la matriz es igual a: %d", determinante); printf("\n\n");

    /*

        Finalmente se procede a determinar si la matriz es regular o no.

        Recordando la teoria de Algebra Lineal, una matriz es regular y tiene inversa si
        su determinante es distinto de cero.

        Por lo que se procede a utilizar un condicional if donde se determina esta condicion.

    */

    if (determinante != 0)
    {
        printf("La matriz es regular y tiene inversa debido a que su determinante es disitinto de 0.\n\n");
    }
    else{
        printf("La matriz no es regular debido a que su determinante es igual a 0.\n\n");
    }


    return 0;
```

Figura 20: Código desarrollado para el problema de Matriz Regular y Determinante

```
Rellenando los valores de la matriz:

Digite un numero
-> Posicion en la matriz [1][1]: 1
Digite un numero
-> Posicion en la matriz [1][2]: 4
Digite un numero
-> Posicion en la matriz [1][3]: 5
Digite un numero
-> Posicion en la matriz [2][1]: 6
Digite un numero
-> Posicion en la matriz [2][2]: 85
Digite un numero
-> Posicion en la matriz [2][3]: 2
Digite un numero
-> Posicion en la matriz [3][1]: 7
Digite un numero
-> Posicion en la matriz [3][2]: 4
Digite un numero
-> Posicion en la matriz [3][3]: 6

Mostrando la Matriz introducida:

 1  4  5
 6  85  2
 7  4  6

El determinante de la matriz es igual a: -2441

La matriz es regular y tiene inversa debido a que su determinante es disitinto de 0.
```

Figura 21: Código en ejecución

En la Figura 22. se puede apreciar el Diagrama de Flujos del programa.

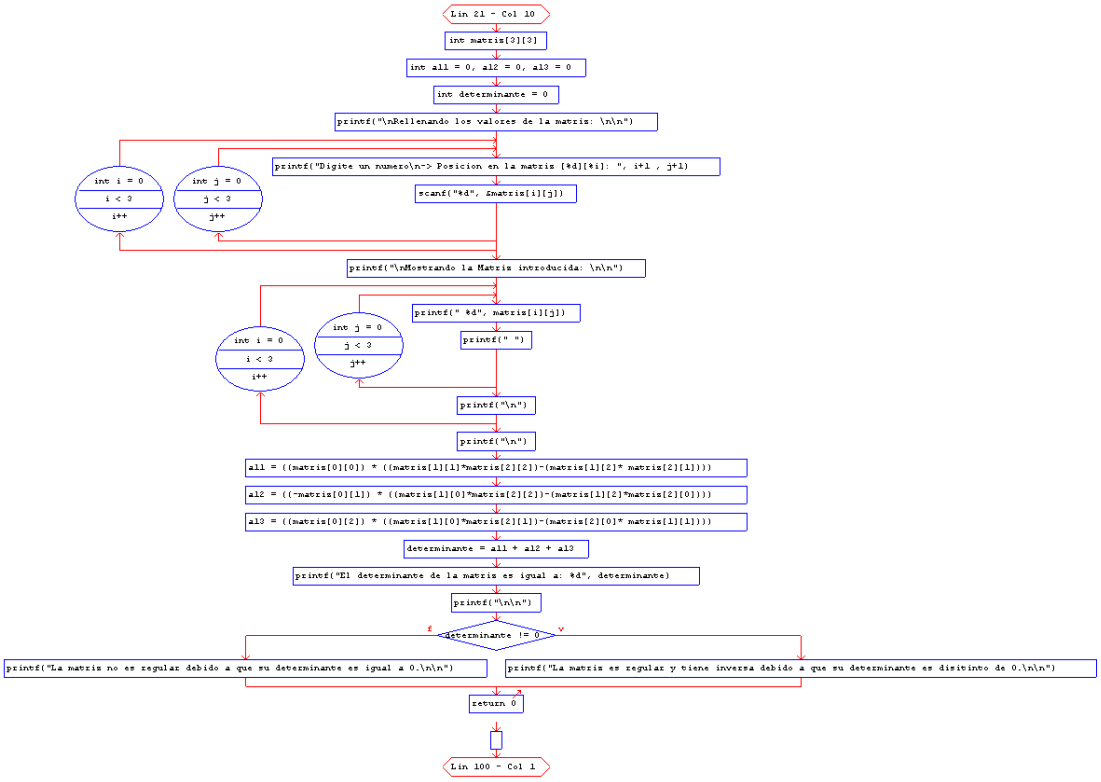


Figura 22: Diagrama de flujo para el problema de Matriz Regular y Determinante

4. Conclusiones y Recomendaciones

La realización de este laboratorio fue de gran importancia debido a que se logró utilizar el lenguaje de programación C ya de una manera más aplicada. Poder aplicar los conceptos de teoría aprendidos en clases permite un entendimiento total de la materia.

Además, el uso de otros recursos como Git y Doxygen para la programación son de gran valor ya que son herramientas esenciales para un futuro si se desea seguir desarrollándose como profesional en el mundo de la programación. Estos recursos permiten un mejor flujo de trabajo más y la facilidad de poder trabajar a tiempo real en el mismo código con varias personas.

5. Referencias

[1] Content, R. (2021). ¿Qué es un lenguaje de programación y qué tipos existen?. Rock Content - ES. Retrieved 11 October 2021, from https://rockcontent.com/es/blog/que-es-un-lenguaje-de-programacion/.

[2] Kernighan, B. W., Ritchie, D. M. (1991). El lenguaje de programación C. Editorial Pearson Educación.

[3] Qué es Git. Atlassian. (2021). Retrieved 11 October 2021, from https://www.atlassian.com/es/git/tutoria is-git.

[4] Getting Started with Doxygen. Cs.smu.ca. (2021). Retrieved 11 October 2021, from https://cs.smu.ca/ porter/csc/common$_3$41$_3$42/$doxygen/doxygen_info.html$.